VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM POLOAUTOMATIZOVANÉ SEGMENTACE ONLINE PÍSMA
SEMI-AUTOMATIC COMPUTERIZED SYSTEM FOR THE SEGMENTATION OF ONLINE HANDWRITING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Michal Gavenčiak**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vojtěch Zvončák**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Michal Gavenčiak ***ID:*** 183430
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Systém poloautomatizované segmentace online písma

POKYNY PRO VYPRACOVÁNÍ:

V rámci práce bude vytvořeno grafické uživatelské rozhraní, které umožní segmentovat jednotlivá cvičení protokolu pro vyšetření dětské dysgrafie, která byla zaznamenána digitalizačním tabletem. Vytvořená aplikace bude obsahovat následné funkcionality: načítání dat písma ve formátu SVC, automatická identifikace časové pozice cvičení v protokolu na základě analýzy digitalizovaného písma, vizuální kontrola cvičení a jejich jednoduché pojmenování. Aplikace bude také umožňovat exportování segmentovaných cvičení do souborů SVC.

DOPORUČENÁ LITERATURA:

[1] J. Mekyska, M. Faundez-Zanuy, Z. Mzourek, Z. Galaz, Z. Smekal, a S. Rosenblum, „Identification and Rating of Developmental Dysgraphia by Handwriting Analysis", IEEE Transactions on Human-Machine Systems, roč. 47, č. 2, s. 235–248, dub. 2017.

[2] Zvončák, V.; Šafárová, K.; Mekyska, J.; Mucha, J.; Kiska, T.; Losenická, B.; Čechová, B.; Francová, P.; Smékal, Z. Automatizovaná diagnóza vývojové dysgrafie založená na kvantitativní analýze online písma. Elektrorevue - Internetový časopis (http://www.elektrorevue.cz), 2018, roč. 20, č. 2, s. 1-6. ISSN: 1213-1539.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Vojtěch Zvončák
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Vývojová dysgrafie postihuje 10-30% dětí školního věku, přesto v České republice neexistuje objektivní způsob její diagnózy, či určení její závažnosti. Proběhlé studie ukázaly, že lze k automatické diagnóze použít digitální data získaná pomocí digitalizujícího tabletu a stylusu. V rámci probíhající studie jsou získána data online písma obsahující informace o poloze, časové informaci, náklonu, přítlaku a azimutu stylusu. Tato data však nejsou připravena na další analýzu vzhledem k obsahu nespecifikovanému počtu cvičení v souborech. V rámci této práce jsou data analyzována a je navržen a implementován program sloužící k segmentaci těchto dat do celků cvičení, umožňující vizuální kontrolu práce programu.

Klíčová slova

GUI, online písmo, Python, segmentace dat, vývojová dysgrafie

Abstract

The prevalence of developmental dysgraphia among school children is between 10-30%, yet in Czech Republic, there is no objective method to diagnose it or determine its severity. Past studies have shown the possibility of automatic diagnosis using digital data gathered using a digitizing tablet and a stylus. Data gathered within an ongoing study contain information on position, time stamp, tilt, pressure and azimuth of the stylus. These data are, however, unsuitable for further analysis due unspecified number of exercises contained in one SVC file. Within this thesis the data is analysed and a program, which is able to segment these data into units of exercises and display the processed data on the screen, is designed and implemented.

Keywords

Data segmentation, developmental dysgraphia, GUI, online handwriting, Python

Bibliografická citace:

GAVENČIAK, Michal. *Systém poloautomatizované segmentace online písma* [online]. Brno, 2019 [cit. 2019-05-25]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/118114. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Vojtěch Zvončák.

Prohlášení

*Prohlašuji, že svou bakalářskou práci na téma „*Systém polo automatizované segmentace online písma*“ jsem vypracoval(-a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(-a) autorská práva třetích osob, zejména jsem nezasáhl(-a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(-a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.*

V Brně dne: 24.Května 2019 ……

podpis autora

Obsah

Seznam obrázků

1. ÚVOD

Potíže s psaným projevem, čí dysgrafie, se vyskytuje u 10-30 %[3] dětí školního věku. Diagnóza této poruchy je však problematická a v současné době v České republice neexistuje objektivní způsob určení její závažnosti, či samotné diagnózy. Nedávno provedená studie [1] však poukazuje na možnost automatizované diagnózy vývojové dysgrafie s použitím analýzy online písma.

U dětí, které se zúčastnily probíhající studie, byla hodnocena závažnost dysgrafie pomocí dotazníku HPSQ (Handwriting proficiency screening questionare). [1] Dále jejich úkolem bylo napsat 3 cvičení psaného textu spojovaným písmem. Děti během vyšetření psaly na papír, který je umístěn na ploše digitalizačního tabletu. Ten převede pohyb speciálního elektronického pera na digitální data s časovou informací.

Pro tuto bakalářskou práci jsou předmětem data, získána stejným způsobem, které se podrobují analýze. Jejich obsahem však není psaný text, ale množina jednoduchých tvarů (spirála, vlnovka, lodička apod.) a obrazců, které dítě dle předlohy kreslí.

Tato data však nejsou ve své prvotní podobě vhodná k analytickému zpracování a vyžadují úpravu. Úkolem této bakalářské práce je navrhnout a vyvinout program, který přípravu dat k další analýze usnadní, či úplně automatizuje.

Pro přesnou analýzu je nutné přistupovat ke každému cvičení individuálně. Soubory protokolů však mnohdy obsahují více než právě jedno. Zároveň však není možné cvičení jednoduše manuálně rozdělit, bez použití vykreslovacího softwaru.

Cílem této práce tedy je vyvinout program, který analyzuje soubor obsahující data z vyplněných protokolů, určí, kolik různých cvičení je v souboru obsaženo, které tahy perem jsou součástí jednotlivých cvičení a následně je rozdělí a uloží do jednotlivých souborů.

Program by měl nabídnout možnosti ovládání pomocí grafického rozhraní, které umožní možnost náhledu na práci programu a případnou korekci automatické segmentace.

2. TEORETICKÝ ROZBOR

2.1 Data

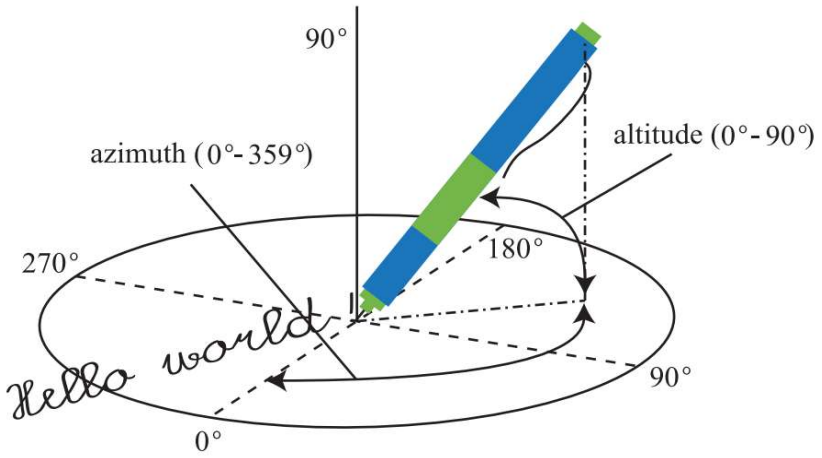
2.1.1 Charakter zpracovávaných dat

Data poskytnutá jako vzor pro vývoj programu jsou uspořádána do celkem dvaceti souborů SVC. Každý soubor obsahuje údaje třízené do sloupců, přičemž každý řádek reprezentuje určitý moment v čase. Každý sloupec reprezentuje vždy jen jeden údaj. Těmito údaji a v tomto pořadí v souboru, jsou:

1. Souřadnice polohy digitálního pera na ose X
2. Souřadnice polohy digitálního pera na ose Y
3. Časový údaj
4. Stav digitálního pera on-surface/in-air
5. Tlak
6. Sklon digitálního pera vůči podložce
7. Azimut sklonu

Tyto soubory se různí svou velikostí, tedy počtem řádků v rozsahu 450 až 36 000 řádků. Obsahem souborů jsou rukou kreslené obrazce různých velikostí a tvarů. Soubor však nespecifikuje počet obrazců, ani jejich polohu, velikost, či tvar. Nejsou rozděleny do sektorů dle souřadnicového systému a vzhledem k problematickému získávání těchto dat, nelze obrazce oddělit podle časového údaje.

Soubory neobsahují ani metadata, která by poskytovala informaci o použitém zaznamenávacím zařízení, nebo jeho technické parametry, které by mohly být užitečné při zpracovávání dat. Podstatnými informacemi by mohly být údaje o maximálním rozlišení záznamu zařízení, nebo počet stupňů detekce přítlaku stylusu.

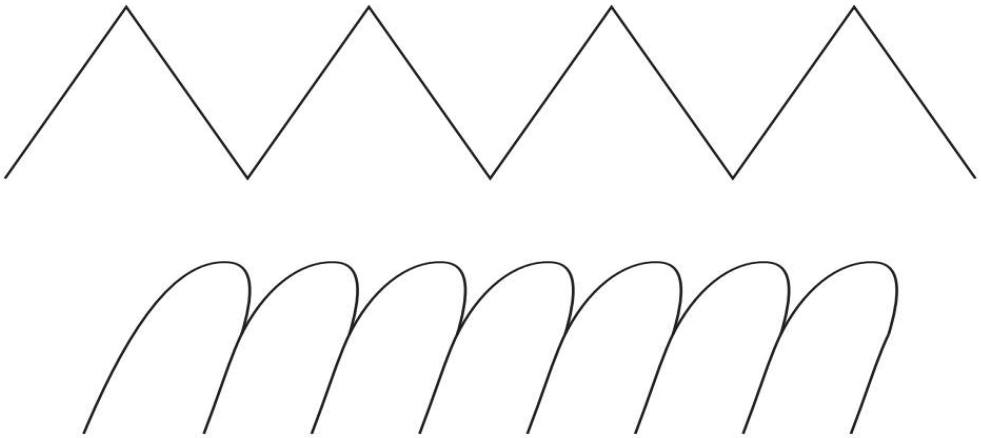


1 - Čtení informací o azimutu, výšce a náklonu [1]

2.1.2 Získání dat

Data byla získána v rámci projektu „18-16835S: Výzkum pokročilých metod diagnózy a hodnocení vývojové dysgrafie založených na kvantitativní analýze online písma a kresby (GAČR)" použitím tabletu Wacom In-tuos Pro L (PTH-850) a stylusu Wacom Inking pen. Frekvence aktualizace stavu digitálního pera byla 150 Hz (vzorkovací perioda T = 6,667 [ms]).

Děti dostaly za úkol nakreslit, dle předlohy, sérii obrazců různících se ve složitosti. Stylus umožňuje použití běžného inkoustu. Toho bylo využito upevněním papíru velikosti A4 na povrch tabletu. Děti tedy kreslily obrazce na papír a měly tak přímou a okamžitou vizuální zpětnou vazbu na své tahy a pociťovaly reálný tlak hrotu a odpor tahů po papíře, který je výrazně vyšší, než při psaní stylusem na povrch displaye.



2 - Ukázka jednoho ze cvičení [1]

2.2 Zvolené prostředky pro implementaci

2.2.1 Programovací jazyk

Pro implementaci programu byly zváženy především dva programovací jazyky; C++ a Python. Výběr jazyka byl proveden především na základě tří parametrů:

- Rychlost běhu implementovaného programu
- Množství a dostupnost nástrojů či knihoven každého jazyka
- Složitost samotné implementace a nutné množství kódu

C++ je multi paradigmatický, staticky typovaný, middle-level jazyk, kompilovaný do platformě specifického strojového kódu, tyto překladače však fungují na velkém množství platforem. Je tedy možné stejný kód zkompilovat pro chod na jiných platformách. Jazyk umožňuje využití ukazatelů a přímého přístupu k paměti. Jazyk je postaven na jazyce C, což do jisté míry zaručuje zpětnou kompatibilitu s tímto jazykem.

Python je objektově orientovaný, dynamicky typovaný, interpretovaný, high-level jazyk. Kód je velice lehce přenosný mezi platformami, bez výrazných úprav, pokud však pro cílovou platformu existuje interpretační program. Ten běží na platformě v pozadí a interpretuje zdrojový kód na strojové instrukce pro danou platformu. Díky nativnímu garbage collectoru není nutná přímá kontrola nad využívanou pamětí, či manuální uvolňování už nepoužívané paměti, usnadňujíc tak složitost implementace kódu za cenu menší kontroly nad výpočetními prostředky přístroje.

2.2.1.1 Srovnání jazyků C++ a Python

Vlastnosti důležité pro mou konkrétní aplikaci se dají shrnout takto:

- Implementace programu v jazyce Python zabere přibližně polovinu času jako implementace v jazyce C++ a výsledný kód má přibližně jen poloviční délku [4]
- Spotřeba dostupné paměti jazyka Python je jen přibližně dvojnásobná oproti jazyku C++ [4]
- Inicializace dat ze souboru může být pětkrát až desetkrát rychlejší u programu implementovaném v jazyce C++, oproti programu psaném v jazyce Python. Běh samotné hlavní části programu psaném v jazyce C++ může být pak dvakrát až třikrát rychlejší.[4]

2.2.1.2 Volba jazyka

I přes velkou výhodu jazyka C++ v rychlosti chodu programu a potřebné paměti, jsem se rozhodl pro implementaci v jazyce Python, konkrétně verze 3.7.0.. Rozhodujícími faktory byly:

- Aplikace nepracuje s velkým objemem dat, rozdíl ve výpočetní rychlosti a spotřebě paměti je tedy zanedbatelný.
- Doba implementace programu v jazyce Python je rychlejší, což umožňuje lepší využití času a dává více prostoru pro experimentování s různými řešeními
- Dle mé osobní rešerše disponuje Python velkým množstvím lehce dostupných a použitelných knihoven a balíčků, např. pro organizaci dat, zpracování dat, či tvorbu grafického rozhraní atp.
- Kód psaný v jazyce Python je i na první pohled lehce čitelný a pochopitelný.

2.2.2 NumPy

NumPy je knihovna jazyka Python. Je součástí takzvaného SciPy zásobníku (ang. SciPy stack), souboru knihoven jazyka Python, zaměřujících se na vědecké výpočetní úkony. Umožňuje jednoduchou práci s vektory, či s vícerozměrnými maticemi. Implementuje funkce, pomocí kterých lze provádět například filtraci prvků, změnu tvaru matic, sloučení a rozdělení matic, či provádět matematické, logické a numerické operace.[5]

Velikou výhodou knihovny NumPy, pro mou práci, je její implementace polí (array). Ty mají podobu tabulky hodnot stejného typu, které jsou indexovány kladnými celočíselnými indexy (včetně nuly).

Pole implementované v nativních knihovnách Pythonu se však liší. Jejich velikost může být dynamicky změněna a prvky tedy mohou být přidány, nebo odebrány. Hlavním rozdílem však je, že tato pole jsou polem referencí, kde každá z nich ukazuje na daný objekt. Z tohoto plyne jak vyšší náročnost na potřebnou paměť, tak delší přístupový čas ke každé potřebné hodnotě.

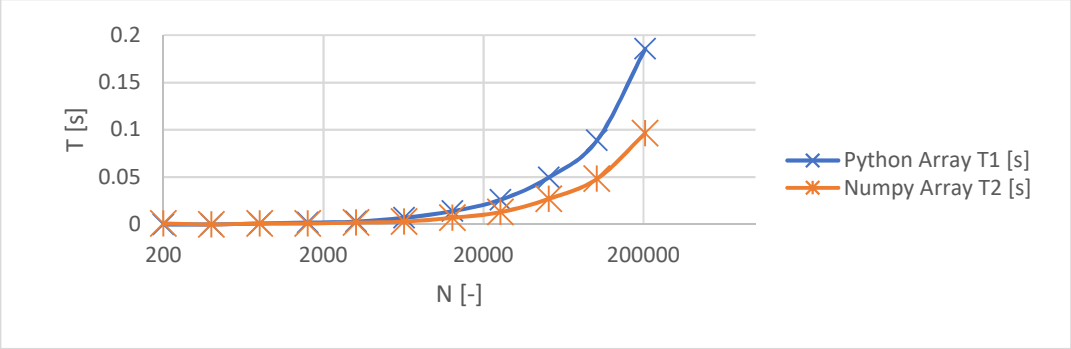
Zavedením jednoduchého kódu v prázdném projektu si můžeme pomocí knihovny NumPy a modulu *time* tyto vlastnosti ověřit. Měření byla provedena taktéž ve verzi Pythonu 3.7.0, distribuce Anaconda 7.2.0, na stolním počítači s operačním systémem Windows 10 (64 bit), procesorem AMD Ryzen 5 2400g a 16 GB operační paměti typu DDR4.

Test je vymyšlen co nejvíce tak aby napodobil operace programu na co nejmenším počtu operací. Nejprve jsou vytvořena dvě pole se stejným počtem seřazených kladných hodnot. Jedno vzestupné, jedno sestupné. Každý i-tý prvek pole, na celém jeho rozsahu, je poté porovnán s i-tým prvkem druhého pole. Pokud je splněna podmínka, je provedena operace. Obě pole jsou pak sečtena do pole nového.

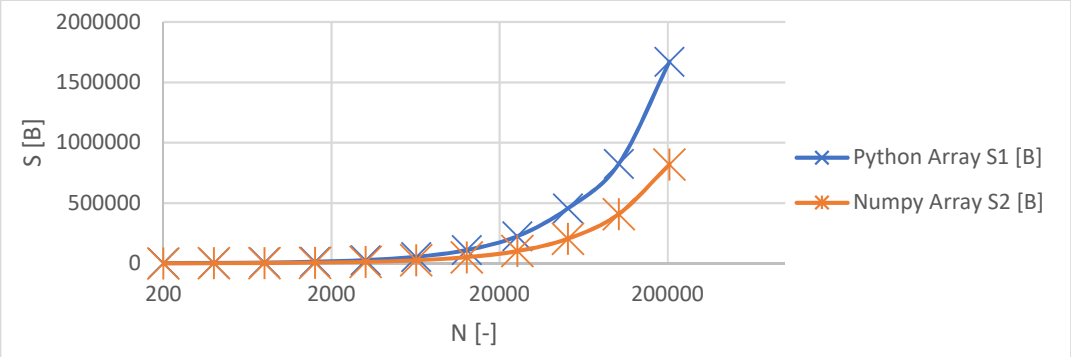
```
size = 30000
start = time.now
X = range(size)
Y = range(size).descending
for i in X:
    if i>Y[i]:
        a = i+Y[i]
Z = X+Y
end = time.now
return (end-start, sizeOf(Z))
```

Pozn.: Jedná se o nastínění kódu pomocí pseudokódu, nikoliv o skutečný zdrojový kód

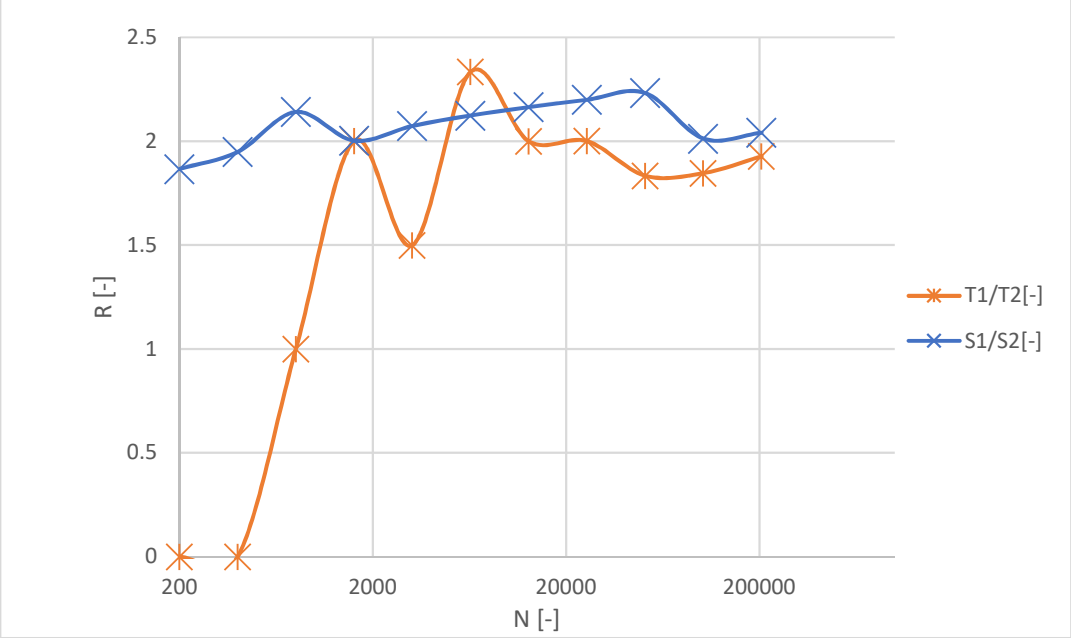
Provedením tohoto testu pro řadu hodnot proměnné *size,* tedy pro několik různých velikostí polí, získáme následující graf ukazující poměr rychlosti a velikosti zabrané paměti pro každou implementaci.



3 - Závislost výpočetního času T [s] na počtu prvků pole N [-]



4 - Závislost velikosti zabrané paměti S [B] na počtu prvků pole N [-]



5 - Závislost poměru R [-] výpočetních časů T1 (Python) ku T2 (NumPy) a velikosti zabrané paměti S1 (Python) ku S2 (NumPy) na počtu prvků pole N [-]

Z grafů je jasně vidět, že zatímco poměr výpočetních časů je ve prospěch polí knihovny NumPy až od velikosti pole N = 1600 [-], velikost zabrané paměti je ve prospěch této knihovny už od velice nízkého počtu vzorků. Je tedy možné vyvodit, že v relevantním rozsahu velikosti pole, což je pro program DysGraphy rozsah N = (800; 50000) [-], je knihovna Numpy přibližně dvakrát rychlejší a zabírá přibližně polovinu paměti.

U počtu prvků polí nižších, než N = 800 [-] nelze určit poměr výpočetních časů, neboť i přes jejich měření na 18 desetinných míst, je naměřena hodnota T = 0,0 [s].

2.2.3 MatPlotLib

MatPlotLib je knihovna jazyka Python, umožňující vykreslování 2D a 3D grafů. Obsahuje funkce pro vykreslení spojnicových, spektrálních, bodových grafů atp. s velkým výběrem stylů zpracování. Podporuje použití ve webových aplikacích i uživatelských grafických rozhraních. [2]

2.2.4 PyQt5

Pro tvorbu grafického rozhraní je použita knihovna PyQt5. Jde o sadu meziplatformních knihoven C++. Umožňuje jednoduchou tvorbu Grafického rozhraní neboli GUI. Nabízí velikou škálu nástrojů, díky kterým lze pracovat například s multimédii, datovými přenosy, kompresí dat, či polohovacími funkcemi.

3. PROGRAM DYSGRAPHY

3.1 Struktura programu

Program je rozdělen do čtyř souborů obsahujících zdrojový kód, Dysgraphy.py, GUI.py, Classes.py a TraceData.py.

DysGraphy.py obsahuje hlavní funkci programu main(). V rámci funkce main() se, v nynější podobě programu, neprovádí žádné výpočetní ani logické operace, dochází pouze ke spuštění grafického rozhraní.

Classes.py obsahuje definice menších tříd (Stroke, OffStroke a MyListWidget), které povětšinou slouží pouze k zapouzdření dat.

GUI.py v sobě skrývá třídu App, definující uživatelské rozhraní.

TraceData.py je v určitém smyslu nejdůležitějším souborem pro chod programu. I když by program nešel spustit ani bez jednoho z nich, tento soubor obsahuje třídy TraceData a Picto, které zapouzdřují nejvitálnější data a definují výpočetní metody.

3.1.1 Třída TraceData

Tato třída se dá považovat za hlavní v rámci funkce programu. Zapouzdřuje proměnné, které obsahují data přímo načtená ze souborů SVC. Dále obsahuje proměnné, které se plní až při běhu programu a obsahují data potřebná pro výpočty a jiné operace s původními daty.

Třída dále implementuje metody, které jsou potřebné pro načtení, zpracování a následné uložení zpracovaných dat. V rámci metod této třídy se vytvářejí, či editují instance třídy Picto.

3.1.2 Třída Picto

Tato třída zapouzdřuje proměnné obsahující výsledná data po zpracování a roztřízení třídou TraceData. Neimplementuje metody pro zpracování dat, pouze pro jednoduché operace, jako jsou přidání či odebrání jednotlivých tahů, nebo uložení do souboru. Metody této třídy se téměř výhradně volají prostřednictvím třídy TraceData, která tak slouží jako její ovládací rozhraní.

3.1.3 Třída App

Třída je třídou dědící z PyQt5 třídy *QMainWindow* a ukrývá v sobě kód definující vzhled a vlastnosti grafického uživatelského rozhraní. Sama obsahuje minimum

logických, či výpočetních operací, zapouzdřuje však referenci na aktuální instanci třídy TraceData, jejíž metody volá po aktivaci příslušného ovládacího prvku.

3.1.4 Třída Stroke a OffStroke

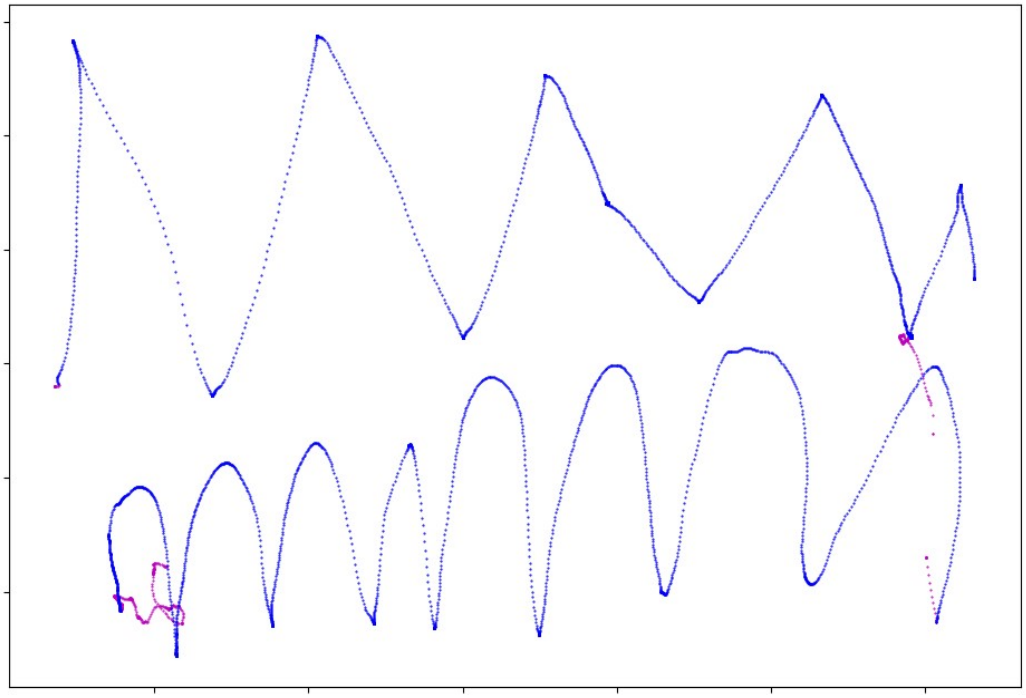
Tyto dvě třídy zapouzdřují proměnné představující jednotlivé tahy a pohyby nad papírem a informace o nich, například jejich minimální a maximální hodnoty, apod.

3.1.5 Třída MyListWidget

Jde o třídu opět dědící z třídy knihovny PyQt5, *QListWidget*, přičemž přidává další zapouzdřená data, ale především obsahuje přetížení metody dropEvent, která je potřebná pro správnou funkci Drag and Drop.

3.2 Diskuse implementace metod segmentace dat

Pro účely této kapitoly budu používat příklady cvičení z vyplněných protokolů. Především ale bude pro ilustraci použito následující cvičení, jehož předloha již byla jednou zobrazena na obrázku číslo 2. Realizováno dítětem, které vyplnilo dotazník, může cvičení vypadat takto.

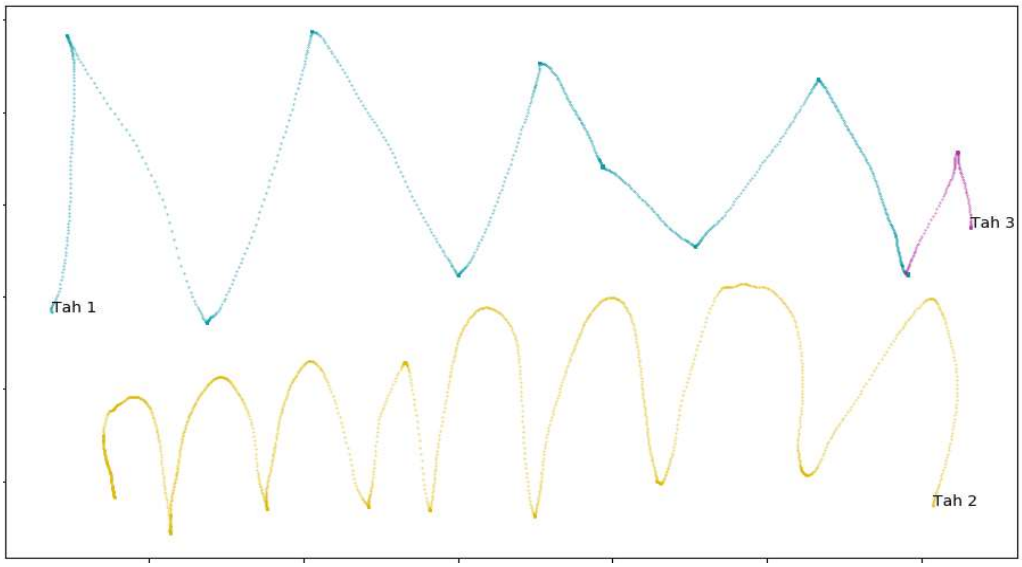


6 - Jedno ze cvičení nakreslené dítětem. Modrá barva - tahy perem na papíře; fialová barva - pohyb perem nad papírem (in-air pohyb)

Vzhledem k tomu, že datové soubory obsahují informace o jednotlivých bodech, bylo nutné jednotlivé body spojit do větších celků, které budou tvořit základní datový prvek. Práce s jednotlivými body by totiž byla, zvláště při manuálním ovládání uživatelem, velice zdlouhavá.

Body se tedy sjednocují do jednotlivých tahů. Toho je docíleno sledováním sloupce stavu stylusu, nosící informaci o on-surface/in-air pohybu. Informace je v podobě číslice 1(on-surface), nebo 0(in-air). Následně je za tah prohlášena každá podmnožina on-surface bodů, která je na svém začátku i konci, vzhledem k časové ose, ohraničena alespoň jedním in-air bodem. Inverzně k tomu jsou hledány in-air pohyby.

Pro účel použití programu se nabízejí různé metody segmentace dat, které lze dle principů jejich exekuce rozdělit do dvou druhů. Metody manuální selekce dat spočívající na vstupu uživatele a metody automatizované segmentace, spočívající na vyhodnocení pomocí analýzy vstupních dat.



7 - Ilustrační cvičení po rozdělení na jednotlivé tahy. Každý tah je reprezentován jinou barvou

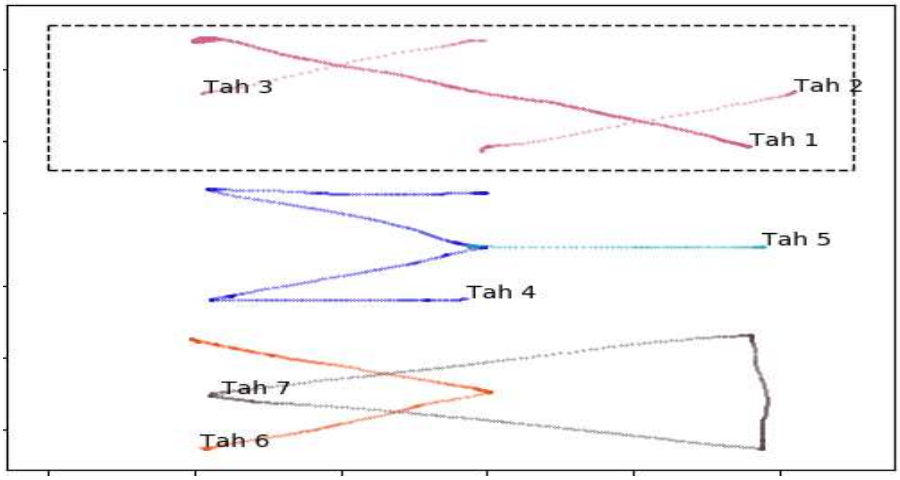
3.2.1 Manuální selekce

Manuální selekce souvisejících tahů je z pohledu složitosti implementace jednodušší než automatická segmentace. Předpokládáme-li zkušeného uživatele, můžeme teoreticky dosáhnout i menšího počtu špatně sjednocených tahů, oproti automatizovanému programu. Nevýhodou manuální selekce je však nutnost lidského, dle určení programu, lékařského personálu, jehož čas by mohl být využit efektivněji. Ruční práce je navíc pro tento druh úkolu standartně pomalejší, oproti práci počítače.

Z možných metod manuální selekce dat se jako perspektivní jeví především dvě metody, závislé na vykreslení dat na obrazovku počítače.

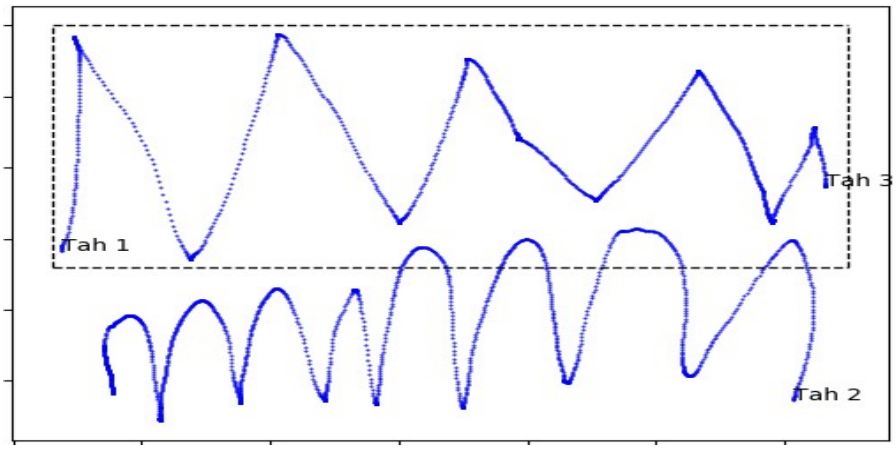
3.2.1.1 Selekce na základě vybrané plochy grafu

Jednou z možností manuální selekce dat je výběr plochy grafu, na které se nachází žádaná podmnožina tahů.



8 - Tahy ve výběru (přerušovaný obdélník) jsou sjednoceny do jednoho obrazce, ostatní zůstávají bez rozdílu

Tato metoda je z metod manuální selekce pravděpodobně nejrychlejší a uživatelsky nenáročná. Pro její efektivitu je však nutné, aby byla cvičení oddělena volnou plochou. Na původním ilustračním cvičení je zřejmý problém použití této metody.

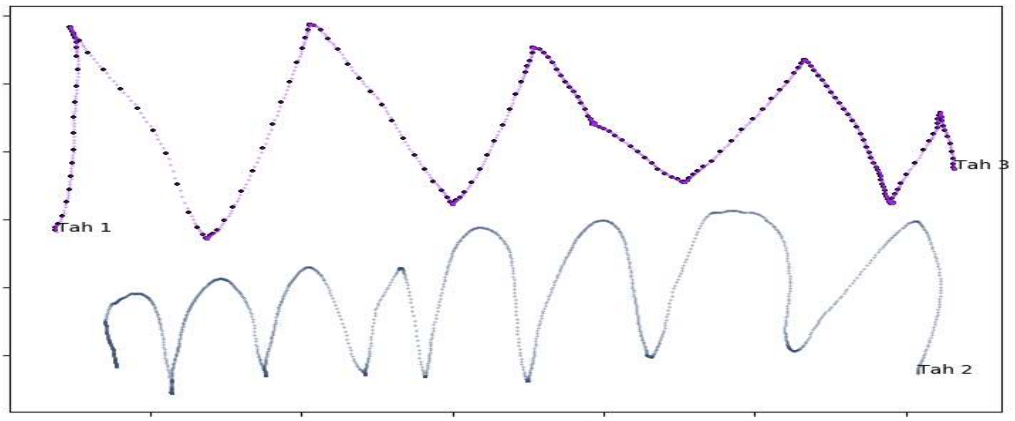


9 - Při výběru jen části tahu není jasné, co se má stát

V tomto případě by ještě nebylo těžké blíže specifikovat podmínky výběru, například tak, aby se do výběru přidaly pouze ty tahy, které jsou vybrány celé. Problém by se však stále mohl vyskytnout v případě, kdy dítě rozdělí jedno cvičení na příliš velké množství tahů. Kdyby například jeden z vrcholů Tahu 2 byl samostatným tahem, stal by se také součástí výše umístěného cvičení, což je zjevně chybný závěr.

3.2.1.2 Selekce označením jednotlivých tahů

Druhou možností manuální selekce je přímý výběr jednotlivých tahů z grafu, nebo vypsaného seznamu, myší.



10 - Vybrané tahy jsou zvýrazněný černou barvou a sjednoceny do jednoho obrazce

Tato metoda je perspektivní díky své přesnosti. Je však velmi zdlouhavá. Jednak kvůli nutnosti zvolení jednotlivých tahů, kterých je zvláště ve větších cvičeních až stovka, ale také kvůli nutnosti bližšího seznámení uživatele s vykresleným dokumentem.

V některých cvičeních se navíc objevují skupinky těsně napasovaných tahů na extrémně malé ploše, kterých můžou být i desítky. Ručně vybrat takovéto jednotlivé mikro tahy je velice zdlouhavé a neefektivní.

3.2.2 Automatická segmentace

Implementačně složitější, ale nakonec časově efektivnější metodou je automatická segmentace samotným programem. Nabízí se více než jedna možnost implementace, včetně využití strojového učení. K té je však potřeba mít přístup k velkému množství vzorových dat, na kterých se program „učí“. Implementace této metody navíc převyšuje současné schopnosti a znalosti autora této bakalářské práce. Jednodušším způsobem je vyhodnocovat související tahy na základě jejich parametrů, jako jsou vzájemná poloha, velikost a čas jejich vzniku relativně k ostatním tahům.

3.3 Výčet funkcí programu DysGraphy

V této kapitole budou vypsány jednotlivé důležité funkce a metody zdrojového kódu. Nejsou zde vypsány naprosto všechny, jen ty, které lze považovat za zajímavé z hlediska jejich implementace, či výsledku.

Je důležité podotknout, že ne všechny vyčtené funkce, především jde o různé druhy vykreslení, jsou aktuálně napojeny na ovládací prvky GUI. Je tedy možné brát tuto kapitolu jako výčet funkcí pomyslné knihovny DysGraphy. Jejich napojení je velice jednoduché, avšak jsem tyto funkce nepovažoval za nutné pro účel programu.

Program obsahuje základní funkce spojené se svým určením. Jednou z jeho hlavních funkcí je automatická segmentace dat, kterou doprovází menší nutné funkce, jako jsou načtení a uložení dat a vykreslení cvičení s různými parametry.

Tato podkapitola neobsahuje výčet všech funkcí a metod nalezených v programu. Uvedeny jsou zde funkce jako celky, z pohledu fungování programu. Samotnou implementací těchto funkcí se budu blíže zabývat v další kapitole.

3.3.1 Ukládání a načítání dat z disku

Odpovídající metody ve zdrojovém kódu:
TraceData_fileToTraceData( )
TraceData_TraceDataToFile( )
TraceData_generateLabFile( )

Program obsahuje tři metody, volané z grafického rozhraní, které operují s daty na disku.

Pro načtení dat ze souboru SVC slouží metoda *fileToTraceData( ).* Data jsou uložena do paměti programu v rámci instance třídy TraceData. Zároveň je uložen i název a umístění souboru.

Pro ukládání na disk slouží dvě různé metody, generující různé soubory.

TraceDataToFile( ) napodobuje takovou organizaci dat, jako mají vstupní soubory SVC, tedy sloupce představující unikátní parametry (poloha, tlak, atd.) a řádky, kde každý představuje jeden zachycený bod v čase. Metoda vytvoří soubor SVC pro každé nalezené cvičení. Pokud je některé cvičení tvořeno nesouvislými tahy, tedy například tahy 1, 2 a tahy 4, 5, jsou pro toto cvičení vytvořeny dva soubory, každý obsahující jeden celek. Název souboru je generován podle vzoru [*původní název souboru*]_[zvolený typ cvičení]_[*xx*]_[*c*].svc. Pokud není zvolen typ cvičení, do názvu se vloží *UnassignedType* a zapíše se parametr *xx*, kde *xx* představuje pořadí takového exportovaného cvičení začínající od *xx* = 1 a *c* představuje pořadí souborů v rámci rozděleného nesouvislého, rozděleného cvičení ve formátu c = {a; b; c; …; y; z; A; B; …; X; Y; Z}.

generateLabFile( ) vygeneruje soubor stejného názvu jako zdrojový soubor SVC, ale s příponou LAB. Tento soubor obsahuje pouze rozsah časové značky každého nalezeného cvičení a jeho typ, oddělené mezerou, ve formátu:

[počátek] [konec] [zvolený typ cvičení]_[*xx*]_[*c*].

Přičemž proměnné *xx* a *c* představují stejné prvky jako u předchozí metody. Výsledný soubor tedy může vypadat přibližně jako na obrázku číslo 11.

```
1556070503 1556095223 TSK1_spiral
1556102276 1556151459 UnassignedType1
1556158657 1556183959 TSK5_saw
1556197629 1556200342 TSK9_recall_a
1556219887 1556231809 TSK9_recall_b
1556202277 1556210568 UnassignedType2
```

11 - Vygenerovaný LAB soubor

3.3.2 Funkce GUI

Jak již bylo řečeno, třída App zapouzdřující GUI nedefinuje téměř žádné logické metody. Místo toho obsahuje funkce určené k ovládání samotného grafického rozhraní. Ty budou v této podkapitole nyní vypsány a sdruženy do jakýchsi celků. Zmíněny zde nebudou metody typu *draw[Foo]( ),* které volají odpovídající metody třídy TraceData a jsou popsány v odpovídající kapitole. (Pozn. autora: "Foo", někdy také „Bar", v programátorské terminologii představuje tzv. „placeholder name", tedy jméno dočasné, nebo jméno které má být nahrazeno)

3.3.2.1 Metody init( )

Třída obsahuje několik metod *init[Foo]*( ), *například initButtons( ).* Tyto metody jsou volány při spuštění programu a starají se o původní vykreslení grafického rozhraní, inicializace potřebných tříd apod.

3.3.2.2 Metoda drawHighlights( )

Jde o metodu pracující s náhledovým oknem aplikace, která je volána po uživatelské interakci se seznamy zpracovaných dat. Když je zavolána, zkontroluje, které prvky seznamu jsou vybrány a ty zvýrazní v náhledovém okně.

3.3.2.3 Metody update( ) a clear( )

Tyto metody se starají o aktualizaci interaktivních prvků uživatelského rozhraní. Například metoda *updatePictos( )* je volána po provedení operace Drag and Drop a slouží k aktualizaci seznamů aplikace a dat v rámci programu.

Naproti tomu metody typu *clear[Foo]( )* slouží k vrácení stavu rozhraní do původního stavu a jsou občas volány jako součást funkcí *update[Foo]( ),* avšak také například při načtení nového souboru ke zpracování, kdy je z metody *defaultGUI( )* volána metoda *clearAllWidgets( ),* která vymaže všechny seznamy a deaktivuje nepotřebné ovládací prvky.

3.3.2.4 Metody pracující s daty na disku

Posledním typem metod jsou metody pracující, prostřednictvím metod knihovny PyQt5, s daty na uložišti počítače. Metody pro ukládání zpracovaných dat jako *generateLAB( )* jsou v principu triviální, neboť pouze otevírají okno pro volbu umístění souboru a následně volají ukládací metody třídy TraceData, které jsou opět popsány v její sekci.

Naproti tomu metoda načítající data z SVC soboru, *openFileNameDialog( ),* po ověření, že byl zvolen platný soubor, spouští reinicializaci třídy TraceData, vyčištění zbytkových dat a vyprázdnění seznamů, tedy vrácení GUI do původního stavu.

3.3.3 Vykreslení dat na obrazovku počítače

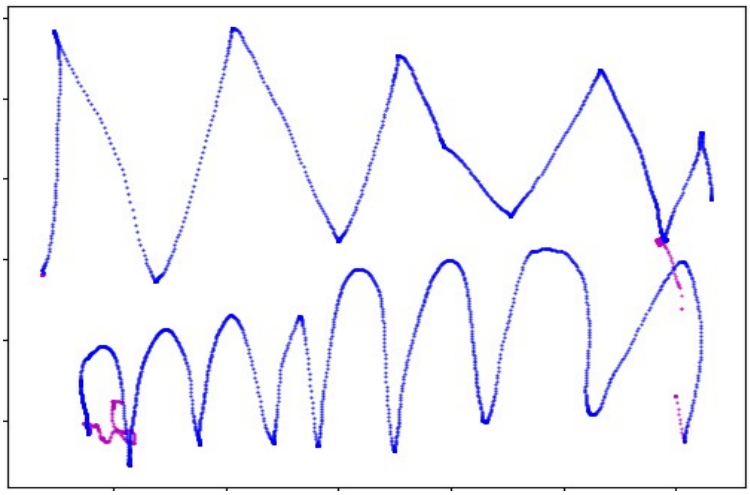
Program obsahuje několik možností vykreslení dat na obrazovku počítače. Každý způsob je unikátní a nese uživateli jiné informace nutné pro vyhodnocení cvičení, či v prvé řadě, ověření správné funkce programu.

Implementace metod zprostředkovávajících vykreslování nebude v rámci textu této práce blíže popsána, neboť jsou v principu velmi triviální a po případném přetřízení dat pouze volají příslušné funkce knihovny MatPlotLib, jejíž dokumentaci lze naleznout na příslušných webových stránkách (https://matplotlib.org/).

3.3.3.1 Vykreslení surových dat

Odpovídající metody ve zdrojovém kódu:

TraceData_drawDefault( )

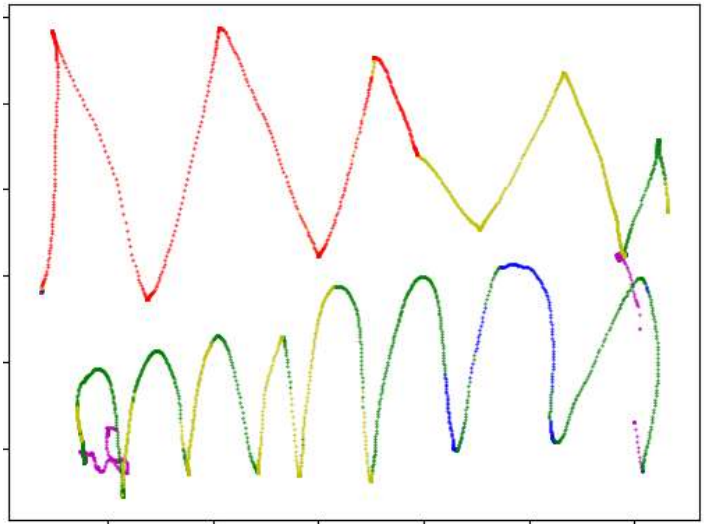


12 - Tahy na papír modře, in-air pohyb fialově

3.3.3.2 Vykreslení dat podle přítlaku

Odpovídající metody ve zdrojovém kódu:
TraceData_renderTracePressurized( )

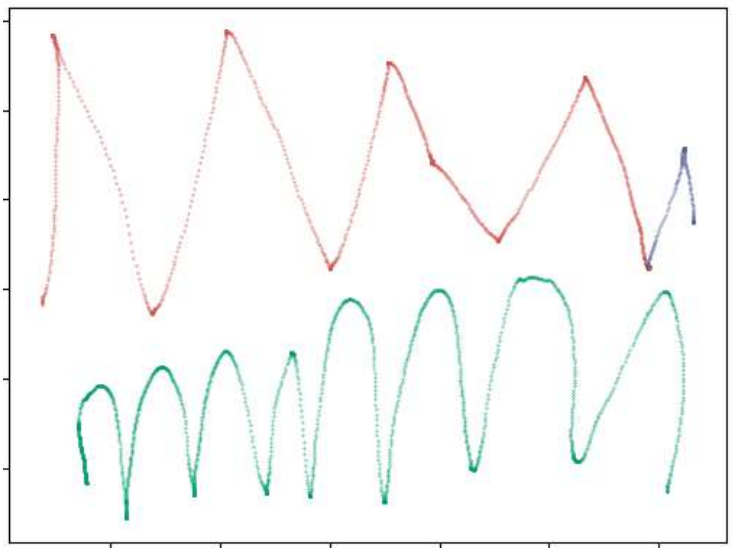
Program vykreslí data a barevně odliší oblasti podle tlaku, které dítě vyvinulo na pero.



13 - Barvy se podle tlaku řadí vzestupně: modrá< zelená< žlutá < červená; fialová: in-air pohyb

3.3.3.3 Vykreslení všech oddělených tahů

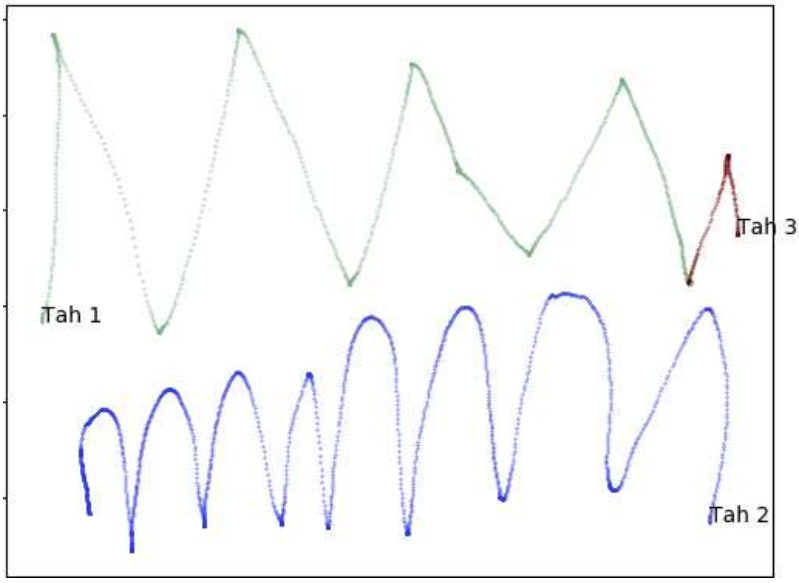
Odpovídající metody ve zdrojovém kódu:
TraceData_renderStrokes( )



14 - Všechny nalezené tahy jsou pro odlišení vykresleny jinou barvou

3.3.3.4 Vykreslení číslování tahů

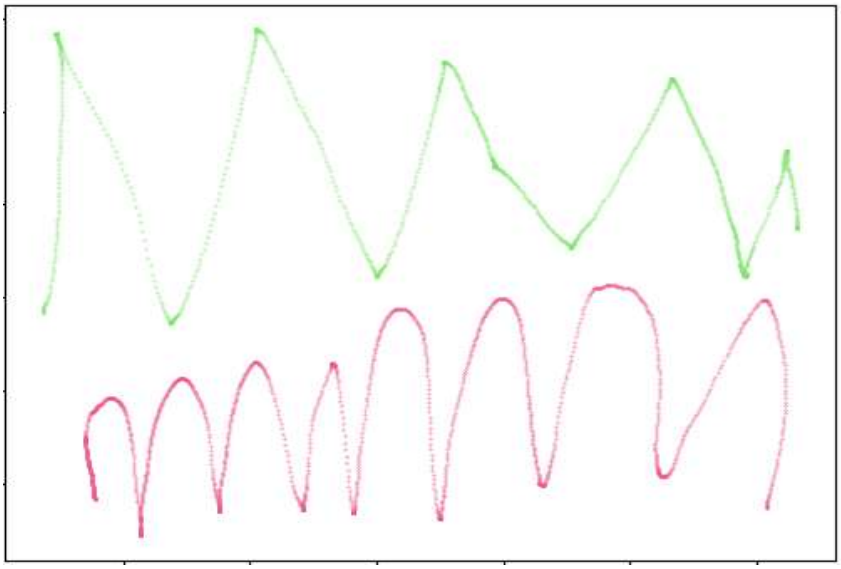
Odpovídající metody ve zdrojovém kódu:
TraceData_renderStrokeNumbers( )



15 - Všechny tahy jsou očíslovány dle pozice na časové ose

3.3.3.5 Vykreslení segmentovaných obrazců

Odpovídající metody ve zdrojovém kódu:
TraceData_drawPictos( )



16 - Všechny nalezené obrazce jsou pro odlišení vykresleny jinou barvou

3.3.4 Segmentace dat

Odpovídající metody ve zdrojovém kódu:
TraceData_translate( )
TraceData_divideStrokesOffStrokes( )
TraceData_sortStrokes( )
TraceData_distributeStrokes( )
TraceData_crossCheckPictos( )
TraceData_assignOffTrace( )
Výčet metod není kompletní. Vypsané metody jsou větší celky, v rámci kterých se provádějí náročnější operace. Jejich průběh detailněji popíšu v rámci této a příštích podkapitol včetně některých zmíněných menších funkcí.

Po načtení potřebných dat, program provede jejich analýzu a v několika krocích je segmentuje. Nejprve na jednotlivé tahy, ty pak sjednotí do celků, formujících obrazce. Data vzniklá v mezikrocích jsou uschována v paměti programu až do jeho případného ukončení. Zpracované obrazce jsou rozděleny do jednotlivých instancí třídy Picto, která umožňuje další operace, především jejich uložení do souboru.

3.4 Algoritmus segmentace dat

Algoritmus segmentace je rozdělen do několika klíčových kroků, jejichž pořadí je nezaměnitelné. V této kapitole se pokusím pomocí příkladů a útržků pseudokódu přiblížit princip funkce programu.

V této kapitole nebude brána v potaz existence GUI a tedy nutná interakce uživatele. Algoritmus bude popsán tak, jako by uživatel postupoval přímočaře od spuštění programu, přes načtení dat, jejich zpracování až po jejich konečné exportování. V pseudokódu však poznačím pozice, kde je interakce s GUI očekávána.

I přesto, že zvoleným jazykem práce je čeština, ukázky pseudokódu budou následovat programovací konvenci, tedy používat angličtinu pro klíčová slova a názvy proměnných, funkcí a metod.

```
Main_Algorithm():
  loadData(„File name“)     #GUI choose file prompt
  drawDefault()             #GUI
  translate()
  divideStrokesOffStrokes()
  sortStrokes()
  distributeStrokes()
  crossCheckPictos()
  assignOffTrace()
  drawPictos()              #GUI
  switch:                   #GUI choose file prompt
    1: saveToSVC:
        traceDataToFile()
    2: generateLabFile:
        generateLabFile()
```

Algoritmus samotné segmentace v tomto příkladu začíná voláním metody *divideStrokesOffStrokes( )* a končí po ukončení metody *assignOffTrace( )* .

3.4.1 Citlivost

Třída TraceData zapouzdřuje proměnnou *(int)proximtyConst*, která je defaultně nastavená na proximityConst = 500. Její hodnota vyjadřuje citlivost detekce blízkých tahů. Díky vysokému rozlišení použitého tabletu nejsou všechny tahy dotaženy tak, aby se skutečně dotýkaly i když to tak na papíře při vyplňování protokolu mohlo vypadat. Samozřejmě není možné vyloučit i naprosté nedotažení tahu.

Například při vykreslení pomyslného písmene „T“ se může jevit, že se nožička písmene skutečně dotýká stříšky, avšak při dostatečném přiblížení může být zřejmé, že nejbližší body těchto dvou tahů od sebe mohou být vzdáleny i stovky bodů vzdálenosti.

Hodnota proximityConst, nebo její násobky se v metodách, které detekují blízkost jiného tahu, přičítá, nebo odčítá od polohových souřadnic bodů, podle potřeby programu.

Tato hodnota je zároveň možností ladění programu, která nevyžaduje zásah do samotného kódu, či chodu programu a může tedy být dostupná i uživateli. Je-li program použit na analýzu cvičení a chybně detekuje velké množství tahů jako souvisejících, dostatečné snížení hodnoty proměnné proximityConst toto množství sníží. Naopak, pokud program nevyhodnotí dva jasně související tahy správně, zvýšení této konstanty povede ke zvýšení minimální vzdálenosti, o kterou mohou být body různých tahů vzdáleny, aby byly vyhodnoceny jako související.

Je však nutné podotknout, že změna hodnoty může chybné vyhodnocení i způsobit. Může například nastat situace, kdy zvýšení citlivosti (zvýšení hodnoty proximityConst) opraví vyhodnocení vztahu dvou tahů, avšak způsobí chybné vyhodnocení jiného páru.

Je nutné poukázat i na skutečnost, že hodnota proměnné proximityConst je absolutní hodnota, pevně dána programátorem, případně uživatelem a nevztahuje se k rozlišení

zařízení, na kterém byla data pořízena. Kdyby byla data zaznamenána s použitím digitizéru s výrazně vyšším, či nižším počtem LPI („řádky na palec“ z ang. lines per inch), dojde k výrazné změně počtu prázdných bodů, které se nachází mezi dvěma tahy i když jejich poloha na ploše digitalizačního tabletu byla stejná.

Hodnota byla nastavena pro konkrétní model zařízení, na kterém byla data získána. Při použití s jiným tabletem je možné, že bude potřeba změnit rozsahy citlivosti. Ideálním řešením by bylo zapsat rozlišení digitizéru do každého zdrojového souboru SVC a pracovat s hodnotou proximityConst jako s relativní k tomuto rozlišení. Toto je však pouhá úvaha k optimalizace aplikace. Její implementace by však byla závislá na týmu provádějící získání dat.

3.4.2 divideStrokesOffStrokes( )

Odpovídající metody ve zdrojovém kódu:
TraceData_divideStrokesOffStrokes( )

Program po jednom projde každý řádek zdrojových dat od nejnižší hodnoty časové informace po nejvyšší, přičemž sleduje stavový sloupec dotyku s podložkou. Souvislé řádky s hodnotou „1“ vyhodnotí jako on-surface a každému takovému úseku vytvoří instanci třídy *Stroke*. Obdobně postupuje i pro souvislé úseky s hodnotou „0“, avšak vytváří instance třídy *OffStroke*. Zároveň do tříd ukládá informace o jejich hraničních bodech na časové ose i v souřadnicovém systému a jejich ohraničujících tazích (ohraničující tahy na podložce pro pohyby nad podložkou a naopak).

Všechny instance těchto tříd jsou poté uloženy do odpovídajících kontejnerů, typu *List*, třídy *TraceData, Strokes* a *OffStrokes*.

3.4.3 sortStrokes( )

Odpovídající metody ve zdrojovém kódu:
TraceData_sortStrokes( )

Všechny nalezené tahy (instance třídy *Stroke* v listu *Strokes*) jsou vzájemně porovnány a vyhodnoceny metodou *checkIfRelated( ).* Tahy, pro které metoda vrátí hodnotu *True* jsou poté přidány do listu *relatedStrokes.*

```
sortStrokes():
    eachWithEach():
        if checkIfRelated(Stroke1, Stroke2):
            relatedStrokes.append(Stroke1, Stroke2)
```

Metoda *checkIfRelated( )* vyhodnotí tři podmínky na jejichž základě vrací hodnotu *True*, nebo *False.*

```
checkIfRelated(Stroke1, Stroke2):
    if oneContainsOther():
        return True
    if strokesTooFar():
        return False
    if strokesDoTouch():
        return True
    else:
        return False
```

3.4.3.1 oneContainsOther()

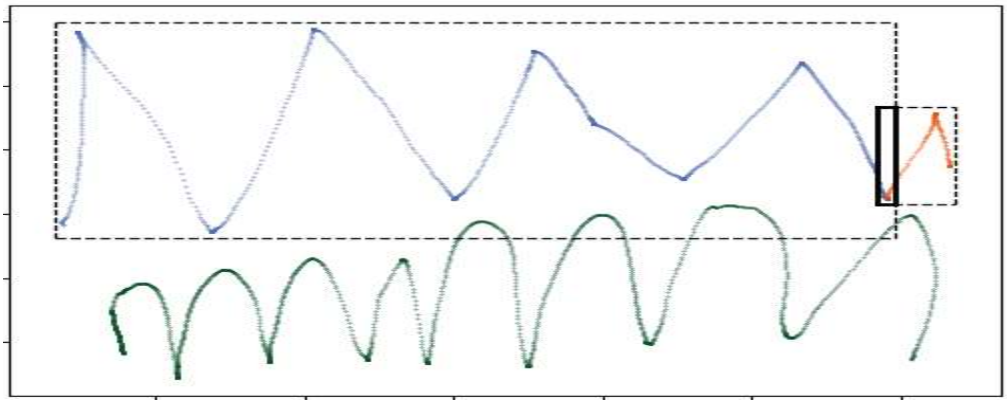
Odpovídající metody ve zdrojovém kódu:
TraceData_contains(stroke1, stroke2)

Metoda najde mezní souřadnice obou tahů, které formují dva obdélníky a porovná je mezi sebou. Pokud je jeden z obdélníků celý obsažen v tom druhém, metoda vrátí True, jinak vrátí mezní souřadnice těchto obdélníků.

3.4.3.2 strokesTooFar( )

Odpovídající metody ve zdrojovém kódu:
TraceData_tooFar(containsResult)

Metoda rozšíří meze tahů o aktuálně nastavenou hodnotu *proximityConst* a prověří, zda se tyto obdélníky protnou. Pokud ne, vrátí False. Pokud ano, vrátí souřadnice sdílené plochy obou obdélníků.



17 - Opsané obdélníky prověřovaných tahů čárkovaně; plocha obsažená v obou obdélnících tlustou čarou

3.4.3.3 strokesDoTouch( )

Odpovídající metody ve zdrojovém kódu:
TraceData_doTouch(stroke1, stroke2, sharedRecX, sharedRecY)

Metoda projde jednotlivé body jednoho z tahů v ploše, která je sdílená oběma tahy. Pokud se ve vzdálenosti, rovné dvojnásobku *proximityConst,* nachází bod druhého tahu, metoda vrátí True.

3.4.4 distributeStrokes( )

Odpovídající metody ve zdrojovém kódu:
TraceData_distributeStrokes( )

Metoda nejprve projde seznam spárovaných tahů a porovná jej se seznamem všech nalezených tahů, zdali byly roztřízeny všechny. Tahy, kterým nebyla nalezena dvojička jsou přidány do *listu Singles*. Každému z nich je poté vytvořena instance třídy *Picto*, do níž jsou poté přidány a ta je přidána do *listu Pictos*.

Poté je prvnímu páru v seznamu spárovaných tahů vytvořena instance třídy *Picto*. Každý další pár je poté kontrolován vůči již existujícím instancím a v ní uloženým rozřazeným tahům. Pokud se jeden z páru kontrolovaných tahů shoduje s již rozřazeným tahem, je ten druhý z páru vložen do stejné instance třídy *Picto*. Pro pár, který nenašel takovouto shodu, je vytvořena další vlastní instance *Picto.*

```
distributeStrokes():
    checkForSingles()
    for single in Singles:
        Pictos.append(new Picto(single))
    Pictos.append(new Picto(relatedStrokes[0]))
    for pair in relatedStrokes:
        added = False
        for p in Pictos:
            if pair[0] or pair[1] matches p[0] or p[1]:
                p.addStroke(uniquePair)
                added = True
        if added == False:
            Pictos.append(new Picto(relatedStrokes[0]))
```

3.4.5 crossCheckPictos( )

Odpovídající metody ve zdrojovém kódu:
TraceData_crossCheckPictos( )

Pokud se obrazec skládá z většího množství tahů (alespoň tří), může se při jejich rozřazování stát, že jsou tahy sjednoceny do více než jedné instance třídy Picto. Zjednodušeně tak můžou existovat dvě instance třídy *Picto*, nazveme je pA a pB, které obsahují tyto podmnožiny tahů: pA = {1; 2; 3}, pB = {3; 4; 5}.

Metoda *crossCheckPictos( )* mezi sebou porovná všechny instance třídy Picto a pokud se ve dvou instancích shoduje alespoň jeden index tahů, jsou voláním metody *mergeToA(pA, pB)* sjednoceny do *pA*. Instance *pB* je poté vyprázdněna a v ní zapouzdřenému stavovému bitu *valid* je přiřazena hodnota *False*.

Poté jsou navíc všechny instance *Picto* porovnány ještě jednou prostřednictvím metody *containsPicto( ),* což je metoda ekvivalentní k metodě *contains( ).* Pokud jsou tedy hraniční souřadnice [X; Y] některé instance zcela uvnitř hraničních souřadnic [X; Y] jiné instance, jsou opět pomocí metody *mergeToA(pA, pB)* sjednoceny do jedné.

3.4.6 assignOffTrace( )

Odpovídající metody ve zdrojovém kódu:
TraceData_assignOffTrace( )

Každé instanci třídy Picto se stavem *valid == True* jsou přiřazeny tahy nad papírem, tak aby každý tah na papíře byl obklopen jedním in-air z každé strany.

3.4.7 Ukončení algoritmu

Běh algoritmu je nyní ukončen. V útržku pseudokódu z kapitoly **3.4** jsou uvedeny ještě kroky uložení dat na disk, závislé na volbách uživatele, ty už se však nedají zahrnout do algoritmu segmentace dat.

3.5 Rychlost a optimalizace algoritmu

V průběhu vývoje programu došlo k několika úpravám kódu, tzv. refactoring. Tyto byly různě rozsáhlé od drobných úprav až po přepsání většiny funkcí v rámci programu. Jedním z důvodů pro úpravy bylo zpřehlednění kódu a uzpůsobení existujících tříd a jejich metod pro snadnější implementaci dalších součástí programu. Tyto vlastnosti již existujícího kódu jsou vitální pro možnost dalšího vývoje.

Další motivací provádět refactoring může být optimalizace chodu programu. Vzhledem k nízkému počtu poskytnutých vzorků cvičení používám pro vyhodnocení rychlosti programu dobu, kterou algoritmus zpracovává jeden z největších dodaných souborů (31830 řádků, 1,127 KB, 10 cvičení, 49 tahů).

Pro měření času byl použit nativní built-in modul jazyka Python, *time*. Měření času je spuštěno před spuštěním segmentačního algoritmu, ale až po načtení dat z disku do paměti programu. Ukončeno je ihned po roztřízení vyhodnocených dat, před vykreslením dat do grafického rozhraní.

```
Data = loadData()
start = time.time()
Solve()
end = time.time()
print(end - start)
GUI.draw()
```

První spolehlivě funkční verze algoritmu, nazvěme ji 1.0, potřebovala na zpracování tohoto souboru 33,60 s. Optimalizace této verze byla zaměřena hlavně na zvýšení efektivity algoritmu, ten fungoval na stejném principu jako teď, ale způsob, kterým fungoval, by se dal označit jako *bruteforce*. Největším skokem vpřed byla právě implementace postupného zpracování dat, v podobě přeskočení tahů, které byly příliš daleko, nebo zúžení rozsahu v rámci tahu, který je důkladně prozkoumán algoritmem. Tento postup odpovídá tomu popsanému v předchozí kapitole, proto jej nebudu opakovaně popisovat. Pomyslná verze 1.1 byla díky těmto úpravám rychlejší o 66,67%. Doba zpracování se tedy snížila na 11,20 s.

Aktuální verze 1.2 vznikla optimalizací zpracování dat v rámci programu. Algoritmus pro správný chod potřebuje informace o tazích, jako jsou hraniční body v rámci časové osy, následující a předchozí tah a pohyb nad papírem, hraniční body tahu v rámci souřadného systému, celkový počet tahů apod. Tyto informace byly z dat získávány v okamžiku, kdy je algoritmus potřeboval, což vedlo k vysokému počtu průchodů daty. Omezením průchodu původních dat na jedno vedlo k dalšímu zrychlení o 55,63%, tedy zkrácení na dobu 4,97 s (zkrácení o 85,21% z původní hodnoty 33,60 s).

Pokud program necháme zpracovat všech 19 poskytnutých vzorových souborů, získáme následující množinu výpočetních dob seřazených od nejmenší: ***T* = {0,07; 0,09; 0,10; 0,12; 0,16; 0,16; 0,21; 0,34; 0,36; 0,70; 0,75; 0,76; 0,82; 1,36; 1,56; 4,97; 7,06; 8,59; 11,86}**

Z této množiny výsledků můžeme vyčíst průměrný čas potřebný ke zpracování souboru, T_P = 2,10 s, přičemž většina (13 z 19) je menší než 1 s a téměř polovina (9 z 19) je nižší, než 0,5 s.

Medián výpočetní doby je roven T_M = 0,70 s.

3.6 Spolehlivost algoritmu a chybná vyhodnocení

Spolehlivost algoritmu lze vyhodnotit na osmnácti poskytnutých souborech (jeden obsahuje ručně psaný text, jehož segmentace není cílem tohoto programu). Je však potřeba podotknout, že takovéto množství dat je poměrně nízké pro správné statistické posouzení přesnosti. Některé z poskytnutých vzorků jsou navíc obrazce, které nejsou obsaženy v předloze cvičení, a proto správnost jejich vyhodnocení nenese důležitou informaci o tom, jak se program chová na skutečném setu dat. Tato podkapitola tedy slouží pouze jako nastínění charakteristik programu.

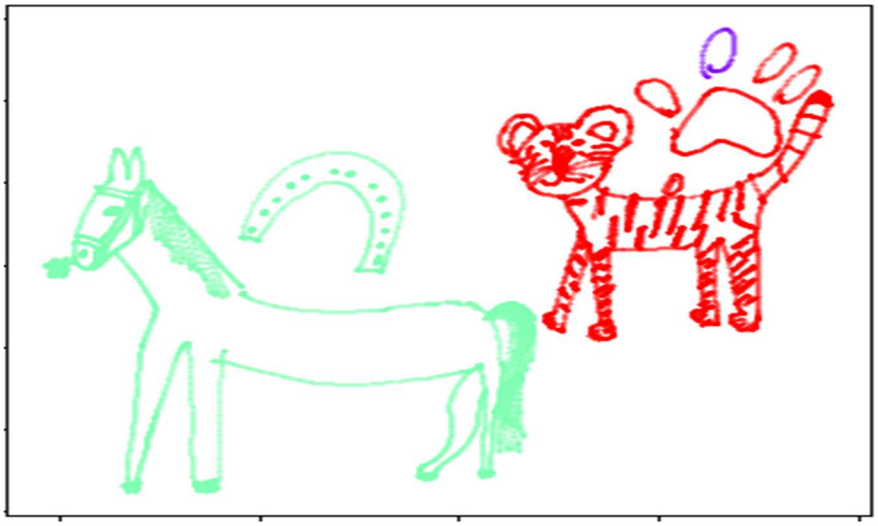
Při defaultním nastavení je správně segmentováno 12 z 18 souborů.

2 z 6 špatně vyhodnocených obsahují obrázky, které nejsou kresleny dle předlohy. Jsou to daleko složitější obrázky (nakreslená zvířata a obrázek explodujících tanků), jež program vyhodnotí téměř správně. Pro naprostou správnost je však nutné využít manuální korekce, kterou program nabízí. Vzhledem k absenci předlohy však není úplně možné určit, jaký stav segmentace je *správný*. (Obrázek č. 14)

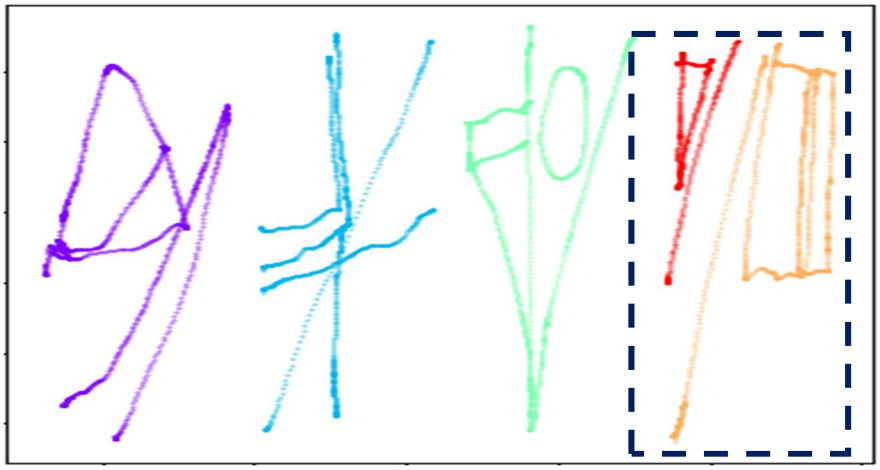
Další 2 soubory obsahují sadu cvičení, při jejichž vyplňování byly tahy umístěny příliš daleko od sebe, algoritmus tedy tyto tahy chybně označí jako různá, unikátní cvičení. Zvýšením nastavení citlivosti algoritmu (S [px]), z původní hodnoty S = 500 px, například na S_I = 600 px dojde ke správnému vyhodnocení obou těchto souborů. (Obrázek č. 15)

Jeden z těchto šesti souborů obsahuje podpis hůlkovým písmem, přičemž není specifikováno, jestli mají být písmena oddělena na jednotlivá, nebo mají být sloučena všechny jako jedno cvičení. Pokud pro účely testování považujeme každé písmeno za samostatné cvičení, dojde ke špatnému vyhodnocení formou sloučení dvou písmen do jednoho cvičení, neboť jsou si příliš blízko. Snížením citlivosti na hodnotu S = 200 px dosáhneme kýženého výsledku. (Obrázek č. 16)

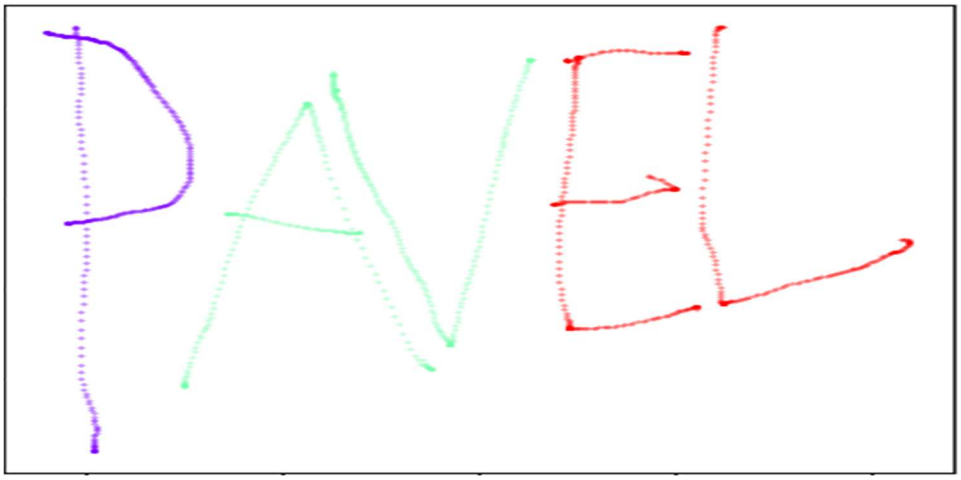
Poslední špatně vyhodnocený soubor obsahuje dvě cvičení, které se výraznou částí protínají, není tedy možné provést korekci pomocí nastavení citlivosti a je nutné ji provést manuálně prostřednictvím grafického rozhraní programu. (Obrázek č. 17)



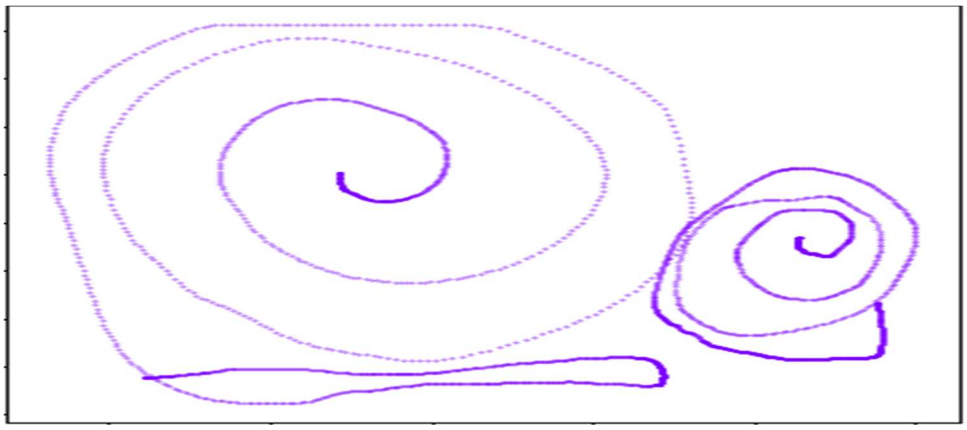
18 - U obrázku kočičky je jeden prst vyhodnocen jako nesouvisející



19 - Dva tahy cvičení napravo jsou seskupeny spolu, už však nejsou seskupeny do celého cvičení (zvýrazněno)



20 - Písmena "A" a "V" a písmena "E" a "L" jsou chybně seskupeny.



21 - Dvě spirály jsou nakresleny přes sebe, což vede ke špatnému vyhodnocení

3.7 GUI a ovládání programu

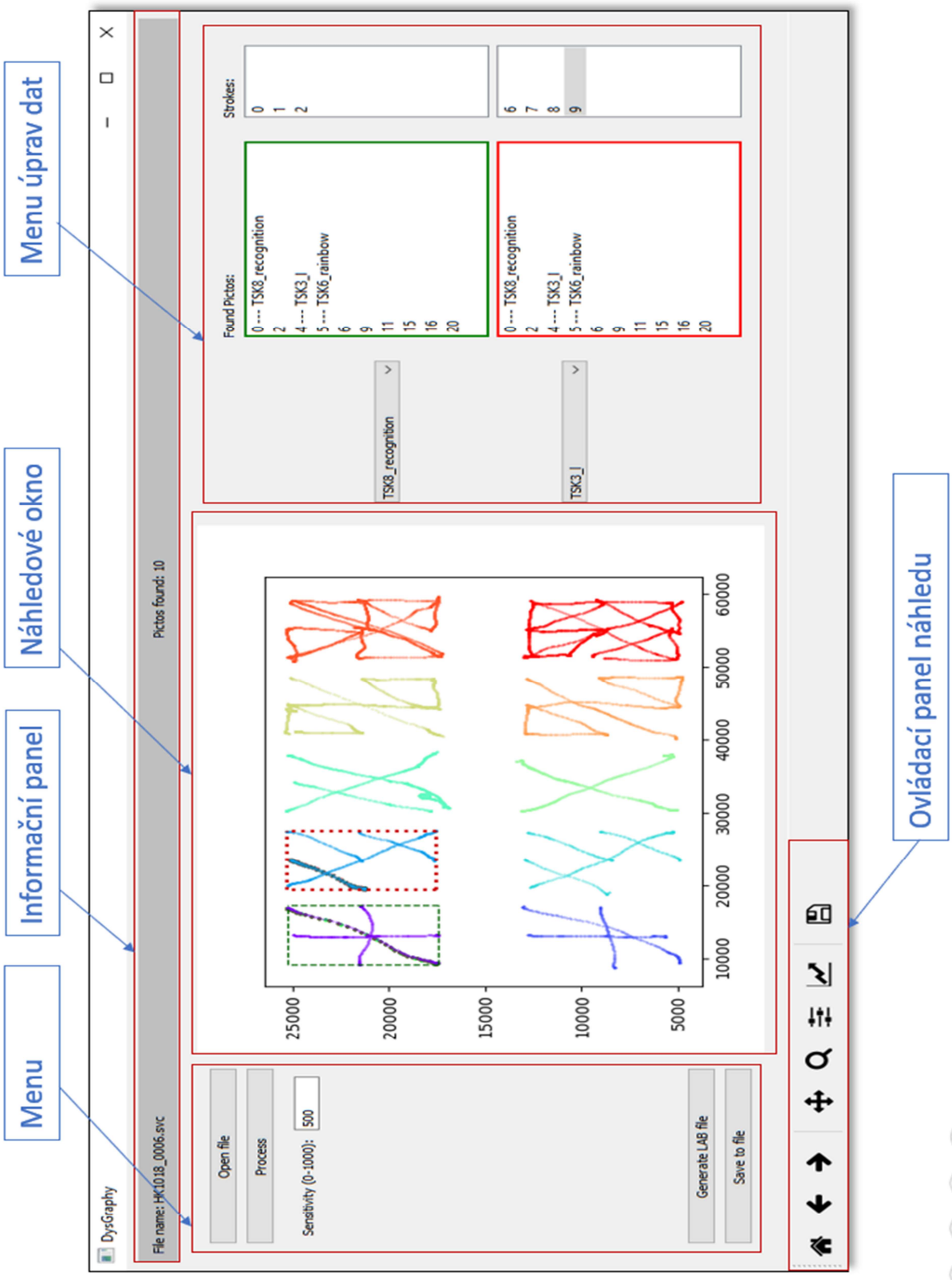
Grafické uživatelské rozhraní využívá knihovnu PyQt 5, která umožňuje jednoduché napojení na knihovnu MatPlotLib, použitou pro vykreslování samotných cvičení. Účelem GUI je nabídnout uživateli kompletní kontrolu nad program a umožní tedy operovat s potřebnými soubory, provádět korekce a nabídne přehledný náhled na zvolený soubor a zpracovaná data. (Obrázek č. 18)

GUI je rozděleno do několika celků, které postupně popíšu, včetně kódové implementace v následujících podkapitolách.

3.7.1 Inicializace programu a GUI

Při spuštění programu je spuštěna inicializace GUI, které je vzápětí zobrazeno. Většina ovládacích prvků je zpočátku neaktivní. Například tlačítko ***Save to File*** se aktivuje až po zpracování dat, tedy ve chvíli, kdy je co ukládat.

Při inicializaci se vytvoří prázdná instance třídy *TraceData* a voláním metody *App_loadPictTypes( )* dojde k načtení seznamu volitelných typů cvičení z textového souboru *PictType.txt*. Jeho název i umístění jsou v kódu pevně dány (hard-coded). Jeho obsah může být tedy změněn, ale vždy musí nést stejné jméno a být ve složce, kde se nachází samotný program.



22 - GUI programu DysGraphy

3.7.2 Menu

Menu nabízí celkem pět položek; tři tlačítka typu *PushButton* a jedno textové pole, neboli *LineEdit*. Funkce jednotlivých položek jsou:

Open File otevře dialogové menu nabízející volbu souboru. Menu se otevře ve složce, ze které je spuštěn samotný program a zároveň je menu ošetřeno na zobrazení a otevírání pouze souborů s příponou .svc. Po zvolení platného souboru jsou obsah, umístění a název souboru načteny do paměti programu. Zároveň je v náhledovém okně vykreslen náhled na nezpracovaná data. Pokud jsou v paměti programu již nějaká data načtena, otevřením nového souboru jsou veškerá data vymazána a následně přepsána.
Tlačítko volá metodu *openFileNameDialog( ).*

Process spustí chod samotného algoritmu. Před jeho započetím je však z pole ***Sensitivity*** načtena zvolená citlivost. Po dokončení chodu programu jsou do náhledového pole zobrazena zpracovaná data.

Sensitivity je textové pole umožňující vložení hodnot citlivosti algoritmu, tedy proměnné *proximityConst*. Do textového pole je umožněno zadat pouze kladné, celočíselné hodnoty v rozsahu $S = [0; 1000]$ *px.*

Generate Lab File - Uživatel je vyzván k volbě složky pro vygenerování souboru s příponou .lab. Po načtení zvoleného umístění dojde k volání metody *TraceData_generateLabFile( )*, která je podrobněji popsána v kapitole 3.3.1.

Save to file Otevře dialogové menu umožňující volbu složky, do které se uloží segmentovaná data. Každé nalezené cvičení se uloží do samostatného souboru, přičemž jeho název je automaticky generován ve tvaru [*původní název souboru*]_[zvolen typ cvičení]_[*xx*].svc, kde *xx* představuje pořadí vytvořeného souboru začínající od $xx = 0$. Pokud není zvolen typ cvičení, do názvu se vloží *UnassignedType*. Podrobnější popis funkce je k nalezení v kapitole 3.3.1.

3.7.3 Informační panel

Informační panel obsahuje základní informace o zvoleném souboru. Ty jsou aktualizovány vždy po stisknutí relevantního tlačítka (tlačítka ***Open File*** a ***Process***). Panel zobrazuje název aktivního souboru a po zpracování dat celkový počet nalezených cvičení.

3.7.4 Náhled na zvolený soubor

Toto pole je vykreslováno knihovnou MatPlotLib. Nabízí náhled na soubor, se kterým se právě pracuje. Náhledové okno je aktualizováno při následujících událostech:

1. Došlo k otevření nového souboru. V tomto případě dojde k vykreslení hrubých dat včetně pohybů nad papírem. (Obrázek č. 8)
2. Došlo ke zpracování souboru. Vykreslí se segmentovaná cvičení, každé jinou barvou, pro přehlednost bez pohybů nad papírem.
3. Došlo ke korekci segmentace

3.7.5 Ovládací panel náhledu

Panel nabízí základní navigaci v rámci náhledového okna. Umožňuje uživateli pohyb v rámci vykreslených dat, zoom, kroky zpět a vpřed, nastavení rozsahu os, uložení do obrázku, a tlačítko *domů*, které vrátí zobrazení do původní polohy a přiblížení.

3.7.6 Menu úprav dat

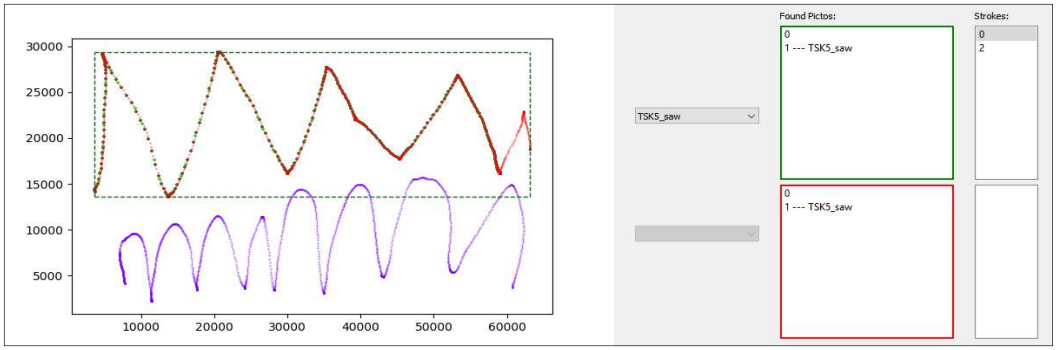
Tato oblast obsahuje celkem čtyři okna, obsahující interaktivní seznamy. Dvě levá okna zobrazují seznam všech nalezených cvičení, přičemž každé je zvýrazněno jinou barvou. Horní zelenou, spodní červenou. Po kliknutí na položku seznamu, představující jedno z nalezených cvičení, dojde k výpisu všech tahů v rámci zvoleného cvičení, v odpovídajícím okně na pravé straně. Zároveň dojde k označení cvičení obdélníkem zobrazeným kolem daného cvičení v náhledovém okně. Obdélník má vždy barvu odpovídající té, kterou je obtaženo okno seznamu.

Seznamy vypsané v oknech pravé strany jsou využity ke korekci chybně vyhodnocených cvičení. Okna podporují funkci Drag and Drop. Pro snazší identifikaci tahu, který je nutné pro korekci přesunout, je po kliknutí na položku seznamu v oknech na pravé straně odpovídající tah zvýrazněn v náhledovém okně.

Zvolením různých nalezených cvičení v oknech na levé straně a přetažením jedné položky mezi okny na pravé straně dojde k přemístění tohoto tahu.

Nalevo od těchto seznamů jsou nalezeny dvě políčka typu ComboBox. Po zvolení cvičení v odpovídajícím seznamu se ComboBox aktivuje. Kliknutím dojde k zobrazení seznamu typů cvičení, ze kterých může uživatel vybrat. Po zvolení požadovaného typu dojde k přejmenování cvičení v rámci seznamu cvičení.

Všechny funkce přejmenování jsou zobrazeny na obrázku číslo 23.



23 - Funkce zvýraznění tahů v rámci náhledového okna a přiřazení typu cvičení

4. ZÁVĚR

V rámci této bakalářské práce byl navržen a implementován program DysGraphy sloužící k segmentaci online písma získaných z protokolů pro diagnostiku vývojové dysgrafie.

Nejprve byl vyvinut algoritmus segmentace, který pomocí polohové a časové analýzy dat segmentuje obsah souboru SVC na jednotlivá cvičení. Algoritmus byl v průběhu vývoje optimalizován, čímž bylo dosaženo zkrácení výpočetního času přibližně o 85%, přičemž průměrná výpočetní doba potřebná k segmentaci souboru je 2,1 s.

Algoritmus byl poté napojen na GUI (grafické uživatelské rozhraní z ang. **G**raphical **U**ser **I**nterface), prostřednictvím kterého může uživatel ovládat program. Mezi nabízenými funkcemi jsou například práce s daty na disku počítače, spouštění a ladění algoritmu, možnost náhledu na hrubá i zpracovaná data a možnost korekce chybně vyhodnocených výsledků algoritmu.

Současná verze programu je připravena k použití ke svému účelu a neobsahuje žádné, vývojáři známé chyby, které by překážely jeho utilizaci.

Z hlediska dalšího vývoje se nabízí především možnosti optimalizace programu, především načítání dat ze souboru a práce s objemným množstvím dat při chodu algoritmu.

Dalším důležitým směrem je vývoj dalších funkcionalit a zdokonalení existujících funkcionalit grafického rozhraní, které, i když funkční, je nepřehledné a neohrabané.

5. CITOVANÁ LITERATURA

[1] MEKYSKA, JIRI, MARCOS FAUNDEZ-ZANUY, ZDENEK MZOUREK, ZOLTAN GALAZ, ZDENEK SMEKAL and SARA ROSENBLUM. Identification and Rating ofDevelopmentalDysgraphia by HandwritingAnalysis. *IEEE Transactions on Human-Machine Systems* [online]. 2017, vol. 47, no. 2, pp. 235-248 [accessed. 13 .December 2018]. Retrieved z: doi:10.1109/thms.2016.2586605

[2] HUNTER, JOHN D. Matplotlib: A 2DGraphics Environment. *Computing in Science &Engineering* [online]. 2007, vol. 9, no. 3, pp. 90-95 [accessed. 13 .December 2018]. Retrieved z: doi:10.1109/mcse.2007.55

[3] KUSHKI, AZADEH, HEIDI SCHWELLNUS, FAIZAH ILYAS and TOM CHAU. Changes in kinetics and kinematicsofhandwritingduring a prolongedwritingtask in childrenwith and withoutdysgraphia. *Research in DevelopmentalDisabilities* [online]. 2011, vol. 32, no. 3, pp. 1058-1064. Retrieved z: doi:10.1016/j.ridd.2011.01.026

[4] PRECHELT, LUTZ. An empiricalcomparisonof C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. 2000.

[5] OLIPHANT, TRAVIS E. *Guide to NumPy*. B.m.: USA: TrelgolPublishing, 2018.